# Chapter 11
# Development Methodologies

**Chapter Overview**

Recent developments in programming theories and technologies present developers with a wider range of tools and techniques. The topics covered in this chapter are designed to give guidance to those projects who choose to use them.

**In This Chapter**

## Section A
## Prototyping

**Section Overview**

A prototype is a working model of the system or application under development.

The scope of the model may range from reproducing or simulating a few selected functions to a fully functional model.

**Caveat**

Even a fully functional prototype by itself is not a complete application, and cannot be put into production without additional follow-on development work.

If it were, it would be defeating the purpose of prototyping which is rapid development of an easily modified model.

**In This Section**

| Topic | See Page |
|---|---|
| Overview | I-11-3 |
| Benefits of Prototyping | I-11-5 |
| Risks of Prototyping | I-11-6 |
| Life Cycle Considerations | I-11-8 |

## Overview

**Definition**

Prototyping is a technique for building a model of an application or parts of an application that allows users and developers to:

- Visualize or define the system
- Refine the system in an iterative fashion
- Reach concurrence on the requirements and interfaces
- Simulate program and process flow
- Provide modifiable code for evolving into the production model

Prototyping is a technique used within the framework of a life cycle.

**When to Use Prototyping**

There are qualities that an application should possess to be a good candidate for prototyping. These include:

- Primarily an on-line transaction-oriented system
- Large multiple record types/relationships
- Willing users and project manager
- Time for iterations/incremental development
- Uncertain/ambiguous requirements
- Difficulty in expressing requirements
- Mainly screen/record manipulation (low algorithmic)
- Report or screen display intensive
- On-Line Analytical Processing (OLAP) systems
- Simple program or data handling logic

*Continued on next page*

## Overview, Continued

**Forms of Prototyping**

Prototyping can take several forms:

- At lower levels, report designers and screen generators can be used to provide users a view of the reports and screens they will see in the final product.
- At higher levels, entire portions of the applications can be created quickly, but without the full functionality of error handling, exception conditions, recovery, security, etc.
- At the highest level, the entire application can be prototyped for user testing and requirements concurrence.

**Types of Prototyping**

There are two types of Prototyping:

- **Evolutionary:** The prototype continues to grow and evolve into a production model as in a Rapid Application Development project.
- **"Throwaway":** A specific objective(s) for the prototype is/are set and once these objectives are met, the model is discarded.

**Data Usage in Prototyping**

The data used to support a prototype may vary from having no data available to using limited or test data.

**Because Customs data are considered "Sensitive but Unclassified", production data (or copies thereof) cannot be used to support prototyping.**

## Benefits of Prototyping

**Introduction**

Prototyping is an iterative process in which tools are used to develop models of part or all of the end product. These models may provide an insight to flaws in proposed system and enable the Systems Development Team to make corrections early in the life cycle.

**Benefits**

There are many benefits that can be derived from prototyping:

- Users can see what is being built for them early in the life cycle and critique it.
- It encourages users to have input into the design process.
- Users may understand and react to prototypes far better than paper specifications. Often they fail to understand, or miss, important points in paper specifications.
- With a good tool, some prototypes can be quickly implemented.
- Prototyping may catch errors and weaknesses before expensive design and programming is done. Modifications or changes to designs are far less costly.
- Prototypes, or partial prototypes, may be of great help in joint application development (JAD) sessions.
- Prototypes may be valuable for communicating to developers what is required.
- Prototypes provide early work experience for users and may be used as a training tool.
- With appropriate software follow-on development, prototypes may evolve into the final system.

## Risks of Prototyping

**Introduction**

Prototyping shares many risks associated with other development methodologies and tools. These risks should be analyzed before a decision to use prototyping is reached.

**Project Risks**

Project risks associated with Prototyping include (but are not necessarily limited to):

- Management may not have a clear picture of actual project progress.
- Configuration control of the numerous iterations can be difficult unless a proper tool is selected.
- Documentation can be weak or non-existent.
- Integration of the iterations may raise problems late in the life cycle.
- Requirements defects may be discovered later in the life cycle, resulting in extensive re-work. “Scope creep” may be endemic.
- Prototyping may or may not allow all individuals involved in the development of the system to see the flaws.
- Results can be misleading if all relevant features are not addressed (i.e., response times during peak usage periods).
- If not required for a specific objective, it can be a waste of resources.

**User-Related Risks**

User-related risk most often center on a lack of self-discipline.

- Users have a tendency to want to turn the prototype into a production system without adequate consideration for security, required audit trails, fallback, recovery, maintainability, performance, networking, or documentation.

*Continued on next page*

## Risks of Prototyping, Continued

**User-Related Risks** (continued)

- Quick and casual design techniques may replace well-structured design practices.
- User expectations may become unrealistic.
- The user may take the prototype too literally when the implemented system may look different.
- The user may not take the time to completely evaluate the prototype, missing potential flaws.

## Life Cycle Considerations

**Introduction**

Prototyping can be used during any or all of the following life cycle phases. Even if prototyping is used in one phase, it need not be used in others.

**Caveat:** Prototyping does not relieve the developer of the need to develop the appropriate SDLC documentation or conduct the appropriate reviews.

**Project Initiation**

Prototyping may aid in the determination of the feasibility of a highly complex project; e.g., “Feasibility Study” or “Proof of Concept”.

**Reference:** Volume I, Chapter 6, *Project Initiation*

**Project Definition**

Prototyping helps during the Project Definition Phase in the following ways:

- Determining System Requirements
- Improving the accuracy of the requirements
- Identifying problems and risks early in the life cycle
- Involving the user in the definition and functional requirements process

**Reference:** Volume II, Chapter 7, Section B, *Project Definition*.

**System Design**

Frequently the prototyping performed in the Project Definition phase also prototypes a portion of the design. Extensive use of prototyping during System Design is most beneficial where CASE tools are available to build the models.

Following completion, the design is baselined in the Critical Design Review.

**Reference:** Volume II, Chapter 7, Section C, *System Design*

**Programming or Construction**

Evolutionary prototypes and partial prototypes can be used to resolve technical issues during Programming or Construction if necessary. If so, results must be documented in the requirements and design as appropriate.

## Section B
## Joint Application Development (JAD)

**Definition**

Joint Application Development (JAD) is defined as a team-based approach to analysis and decision-making. It is a joint venture between customers/users and information systems personnel, centering around a structured workshop, or JAD session.

**In This Section**

## Overview

**Types of JADS**

There are three basic types of JAD:

- Planning JADs are used to:
  - Determine how a project will be organized
  - Prioritize projects or strategic planning
  - Plan multi-site or multi-phased implementations
  - Plan other JADs; i.e., determining the number of JADs needed, the JAD objectives, schedules, participants, etc.
- Single JADs are used to obtain information in a single session, lasting from one to five days. This is the most simple and most widely used form of JAD.

  **Examples:**

  - Defining enhancements to a major system
  - Revisions to an existing document
- Multiple JADs are used for related projects and involve a series of JAD sessions, usually scheduled back-to-back.
  - Output from one session is often used as input into the next session.
  - The same participants are sometimes required for several sessions.
  - Multiple systems or large complex systems with multiple functions are good candidates for this type of JAD.

*Continued on next page*

## Overview, Continued

**When to Use JAD**

Projects that are candidates for JAD can be identified based on:

- **Criticality:** Projects that are influenced by political or policy issues. These projects are usually:
    - ‣ Very large in project size or scope
    - ‣ Highly visible
    - ‣ Critical to the success of the organization's mission
- **Diversity:** Personnel with information, requirements, and needs from multiple organizations are active in these projects. There are often multiple priorities and locations.
- **Risk:** Project risks that can be addressed by JAD include:
    - ‣ Projects with troubled pasts
    - ‣ Projects that have been started and stopped many times
    - ‣ Extensions of a system that has operated poorly in the past
    - ‣ Projects where the relationship between the developers and the user has been very weak
    - ‣ Diverse projects where the various groups of users are in disagreement over the project
- **Time and budget constraints:**
    - ‣ JAD may decrease development time by compressing the process necessary to gather a complete set of user requirements.
    - ‣ JAD may result in better planning, thereby decreasing the amount of re-work that may have to be done later in the project.
- **Communication:** JAD can be helpful in presenting project benefits, goals, objectives, or alternate approaches to those who might not have a clear understanding of them or the overall project prior to the JAD.

## Benefits of JAD

**Introduction**

The benefits of using JAD may include accelerated system design, increased quality of the final product, and improved customer relations.. Using JAD techniques with an independent facilitator in decision-making meetings may also ease the process of gathering a required consensus.

**Consensus**

JAD may be used to reduce many short meetings attended by different groups of people into one workshop attended by everyone involved in the project. This produces the following benefits:

- Group consensus may shorten the traditional approval processes.
- Agreements and commitments may be finalized if the appropriate parties are present.
- Project requirements may be better defined, negating the need for time spent later to:
    - Gather/document additional or overlooked requirements
    - Obtaining clarification from users

**User Involvement**

The JAD methodology encourages user involvement in decisions and user ownership of the results. This may result in:

- A more complete statement of user requirements to be satisfied by the project
- Greater accuracy in the definition of the requirements

**Communication**

JAD helps to bridge the gap between users and information systems developers in the following ways:

- The System Development Team can get information from the users first-hand, reducing the possibility of miscommunication.
- The System Development Team and the users come to understand the other's environment, activities, and overall requirements.

## Risks of Using JAD

**Introduction**

Careful planning is essential to increase the likelihood of realizing the maximum benefits of a JAD session.

**Selecting Qualified Participants**

In order to ensure that the right people are in the session, the following questions must be asked:

- Does this individual know enough about the subject to make a genuine contribution?

- Does this person have the authority to make binding decisions or resource commitments if necessary? If not, am I (as a manager) willing to delegate such authority in this matter?

  If the answer to either of these questions is "no", second thought should be given to assigning that individual to a JAD.

- Does this individual have a personal agenda that might suborn or sabotage the group's efforts or the results?

  If the answer to this question is "yes", the individual should not be assigned to a JAD.

**The Business Sponsor**

Business Sponsor(s) involvement is essential for a successful JAD. Without resource commitments in the form of time, personnel, and funding (when necessary), a JAD session cannot succeed.

**Reasonable Objectives and Expectations**

The appropriateness of the objectives and expectations can be determined by examining the following factors:

- Complexity of the questions to be addressed by the group
- The time allotted in the development schedule for the JAD process
- The presence or absence of key stakeholders and/or their representatives

**Trained Facilitator**

The absence of a well-trained facilitator can have a major, negative impact on the JAD sessions.

## The JAD Process

**Introduction**

Organizing a successful JAD involves three phases

- Pre-session Activities
- Workshop Activities
- Post-session Activities

Each phase is discussed below.

**Pre-Session Activities**

There are five pre-session activities that must be completed:

- **Definition:** During this activity, an accurate assessment of the project determines:
  - The biases and assumptions which may be inherent in the project
  - The viability of using JAD
  - Team roles and players
  - Scope and boundaries of both the project and the JAD
  - The level of management commitment
  - Any relevant background.
- **Research** focuses on the perspective of the customers in order to:
  - Validate findings
  - Clarify pertinent political and cultural issues
  - Address particulars of language and jargon
  - Identify the general business being addressed by the project
- **JAD Deliverable Definition and Data Capture Design** focuses on:
  - Further project definition
  - Designing and developing the means of recording workshop data
  - Workshop deliverables are determined

  **Golden Rule of JAD: If it is not written down or recorded in some agreed-to fashion, it did not happen and does not exist.**

*Continued on next page*

## JAD Process, Continued

**Pre-Session Activities** (continued)

- **Session Structure and Schedule** focuses on producing the:

  - Workshop technical agenda
  - Workshop contents
  - Final schedule

  In addition, scripts are developed and sequencing, numbers, and lengths of workshops are determined.

- **Workshop Readiness:** Here commitment to proceed is established, kickoff meetings occur, and logistics are finalized.

**Workshop Activities**

Workshops are the heart of the JAD process. Each workshop is a team-oriented, consensus-based decision-making process with its own specific technical objectives.

- **General Objectives:** In addition to the technical objectives, each JAD workshop must meet the following general objectives:

  - Identify and resolve critical project issues
  - Achieve consensus through structured exercises
  - Construct a solution documented in a deliverable

- **Participation:** All participants are considered to be equals, with equal ability to influence the final decisions.

- **Issue Resolution/Disposition:** When a question cannot be resolved, it is designated an open issue. Open issues are dealt with in one of three manners:

  - An issue may be resolved through group discussion
  - An issue maybe added to a list of assumptions to be validated later
  - An issue may be added to a list of unresolved issues

  At the end of the session, all assumptions and unresolved issues are assigned to appropriate personnel or subgroups for research, resolution, and/or validation.

*Continued on next page*

## JAD Process, Continued

**Post-Session Activities**

Once the JAD workshop has been completed, the task of finalizing the JAD deliverable begins. This task includes:

- Validation of workshop information
- Resolution of unresolved issues
- Validation of workshop assumptions
- Production of the JAD workshop deliverable
- If appropriate, integration of this JAD workshop deliverable with other JAD workshop deliverables

**Essential Roles and Responsibilities**

There are six essential roles associated with Joint Application Development.

| Role | Responsibilities |
|---|---|
| Facilitator | • Creating and maintaining the process the JAD will use<br>• Creating and maintaining the meeting structure<br>• Leading the discussions<br>• Challenging the group to come to consensus or closure when necessary |

*Continued on next page*

## The JAD Process, Continued

**Essential Roles and Responsibilities** (continued)

| Role | Responsibilities |
|---|---|
| Documentation Specialist | • Assisting with the selection of documentation methods/tools<br>• Documenting the entire JAD process<br>• Preparing an initial draft of the agreed-upon deliverables<br>• Preparing the final draft of the deliverables based on comments from all participants<br>**Note:** If the JAD session consists of more than four or five individuals, it may be advisable to have two recorders. |
| Business Sponsor | • Setting objectives and expectations for the JAD sessions<br>• Ensuring that the System Development Team has access to and commitment from the users<br>• Ensuring that the proper people are available to attend the JAD session<br>• Providing any extra funding (i.e., travel funds, etc) required by the JAD |
| Project Manager | • Delivering of project work products<br>• Coordinating the project effort<br>• Ensuring that strong communication exists between the users and System Development Team |

*Continued on next page*

## The JAD Process, Continued

**Essential Roles and Responsibilities** (continued)

| Role | Responsibilities |
|---|---|
| User | • Participating in the JAD session(s)<br>• Supplying business area knowledge to be used in system development |
| System Development Team | • Participating in the JAD session(s) as appropriate<br>• Ensuring that the system to be developed is realistic<br>• Ensuring that the system required can be delivered when the users need it<br>• Ensuring that the system makes the most effective use of the organization's technological architecture |

**Optional Roles and Responsibilities**

There are two optional roles associated with Joint Application Development.

| Role | Responsibilities |
|---|---|
| Technical Expert | • To provide special knowledge or expertise on a specified range of technical matters<br>• To provide experience-based opinions on technical issues under discussion<br>**Note:** These activities are performed at the express invitation/request of the JAD participants. |
| Observer | • Attending the sessions but not actively participating |

## Life Cycle Considerations

**Introduction**

The JAD process can be adapted to accommodate many types of projects and stages in the life cycle.

**Types of Outputs**

The following are examples of JAD outputs:

- Requirements Specifications
- Systems Modifications
- Re-engineered Business Processes
- Report Layouts
- Screen Designs
- Documentation

**Life Cycle Phases**

- **Project Initiation Phase:** During this phase JAD may contribute to defining the problem or opportunity that the user has identified and planning the project to solve the problem or achieve the customer's goal.

- **Project Definition Phase:** During this phase, JAD helps in determining what the system will do, what functions the system will provide, and the work flow.

- **System Design Phase:** JAD may support how this will be accomplished, including the development of design specifications for data elements, screens and reports. JAD has also been used successfully to define a system's logical database design.

- **Programming Phase:** Technical JADs may be held within the programming environment to resolve issues and challenges.

- **Implementation Phase:** Renewed commitment from the users may be obtained using JAD to update the initial implementation plan.

- **Operations Phase:** Major enhancements or changes to the basic design assumptions may benefit from using a JAD approach.

## Section C
## CASE Tools

**Section Overview**

This section on computer-aided software engineering (CASE) tools will provide the reader with a generic understanding of this software development technology.

**Types of Case Tools**

- **Front-end CASE tools:** Picture drawers or screen painters that use graphics to represent concepts or programming objects
- **Back-end CASE tools:** Code generators that use the power of the computer to derive detailed information from less detailed specifications
- **Stand-alone CASE tools:** Tools that address limited portions of the life cycle
- **Integrated CASE tools:** Tools that automate the entire life cycle with a model providing the framework

**Essential Components**

Two essential components for automated software development are:

- Data dictionaries
- Design dictionaries: which include third, fourth, and fifth generation data (GD) base management systems (DBMSs).

These support the development of data models and procedure models, which are then implemented with the most appropriate software and the most appropriate generation of programming languages (GLs).

**In This Section**

| Topic | See Page |
|---|---|
| Overview | I-11-21 |
| Data Dictionaries | I-11-23 |
| Design Support | I-11-26 |
| Documentation Support | I-11-27 |

## Overview

**Definitions**

**Computer-Aided Software Engineering (CASE):** A generic term that refers to tools that support the automation of software development. It encompasses all stages of the software development life cycle. It is based on a rigorous methodology, with software tools to automate application of that methodology by development staff and users.

**CASE Tools:** An integrated set of software tools and automated programs that simplify application development, improve system quality, and enhance productivity.

**Good CASE Tools**

The best CASE tools provide:

- Database management systems
- Multi-task application managers
- Program management aids
- Decision support systems
- A comprehensive tool chest, including:
  - Document and spreadsheet tools
  - Modeling and object tools
  - Structured Analysis tools
  - Artificial intelligence tools
- Idea processors
- Mathematical programming

*Continued on next page*

## Overview, Continued

**Good CASE Tools** (continued)

- Simulation
- CAD/CAM and Simulation
- Expert Systems
- Languages

**Note:** Not all CASE tool products are fully integrated and seamless.

## Data Dictionaries

**Functional Characteristics of Data Dictionaries**

- Data dictionaries assist in the requirements specification by allowing the CASE tool to automate the required cross-checks.
- A design dictionary may be an expert system for automated analysis, design and development support.
- In their ultimate form, they can automatically generate fully executable systems directly from design specifications prepared by users.

**Dictionary Class Summary**

Data dictionaries and design dictionaries are categorized into four main classes.

| Class | Dictionary Type | Software Type[1] |
|---|---|---|
| 1 | Passive data dictionaries | For 3GD, 3GL and 4GL |
| 2 | Integrated active data dictionaries | For 3GD, 4GD, 3GL and 4GL |
| 3 | Data-driven design dictionaries | For 3GD, 4GD, 3GL and 4GL |
| 4 | User-driven expert design dictionaries | For 3GD, 4GD, 5GD, 3GL, 4GL and 5GL |

[1]**Types:** 3GD = Third Generation DBMS
4GD = Fourth Generation DBMS
5GD = Fifth Generation DBMS
3GL = Third Generation Language
4GL = Fourth Generation Language
5GL = Fifth Generation Language

*Continued on next page*

## Data Dictionaries, Continued

**Characteristics of Class 1 Data Dictionary**

- Class 1 dictionaries provide no active support to the design process.
- They are data administrator and database administrator tools.
- Their support is passively directed to the generation of various database schemata for physical database implementation once data definition and design have been completed.
- They are used primarily for third generation DBMS products and languages, with some limited application for 4th generation languages (GLs).

**Characteristics of Class 2 Data Dictionary**

- Class 2 dictionaries provide more integrated support for 4GLs and for some 3GLs, integrating the languages they support with the third or fourth generation DBMS products on which those languages are implemented.
- Class 2 dictionaries are often proprietary, developed specifically to support proprietary DBMS products and languages.
- They are referred to as ‘active’ dictionaries in their support for database design and implementation.
- Their support for analysis and design is passive; they are wholly dependent on the expertise of analysts.

**Characteristics of Class 3 Data Dictionary**

- Class 3 dictionaries are oriented towards technical personnel.
- They are products developed for use by experienced analysts and technicians as well as data administrators and database administrators.
- They provide assistance in analysis and design, supporting interactive graphical design by experienced systems development staff.
- These design dictionaries record the graphical design documentation for later revision and refinement by data administration staff.
- They are used to design both third and fourth generation databases and systems which use 3GLs and 4GLs.

*Continued on next page*

## Data Dictionaries, Continued

**Characteristics of Class 3 Data Dictionary** (continued)

- Development staff provide the technical expertise, while users are involved only in reviewing the resulting diagrams and designs.
- Class 3 dictionaries cannot support the active user-driven analysis, design and development support.

**Characteristics of Class 4 Data Dictionary**

- Class 4 expert design dictionaries are user driven.
- The expert design support of class 4 dictionaries enables that software to progressively '*learn*' about the system being designed.

## Design Support

**Introduction**

CASE tools support system and process design by supporting both data models and procedure models. If structured analysis and structured design are supported, data flow diagrams and structure charts are also modeled.

**Definitions**

**Data Model:** Comprised of a data map and data definitions documented in an entity list. It provides feedback for identification of information requirements.

**Data Map:** A data map provides a graphical representation of data entities and the associations between them for detailed user review.

A data map and its data definitions directly support each other in a data model.

**Procedure Model:** A procedure model can be derived from a data model. It comprises a procedure map and processing definitions in a procedure list. It documents the logic required to process the data represented by the data model.

**Procedure Map:** A procedure map provides a graphical representation of procedural logic and business conditions to be satisfied. It provides feedback to users for identification of their processing needs.

A procedure map and its process definitions directly support each other in a procedure model.

**Model Integration**

Modeling combines the data model and the procedure model in an iterative process: changes in definitions must be reflected as appropriate in the supporting data and procedures maps, and vice versa.

The CASE tool should automatically integrate a data model and its associated procedure models.

## Documentation Support

**Introduction**

Much documentation is produced throughout the systems development life cycle. This documentation must be maintained so that it is complete, consistent and up-to-date.

**Good CASE Documentation Tools**

- The software automatically produces updated documentation fully consistent with the applied changes.
- Subject databases, implementation clusters and project plans are automatically derived, highlighting the impact of the change.
- Supporting detailed data and strategic plan documentation is automatically produced.
- Changes in priority areas can be readily identified for early development and delivery of systems.
- Procedure maps are then developed from the derived clusters and updated data maps.
- Users can review the business impact of the change and refinements can be made where necessary, leading to further active and automatic maintenance of documentation.
- The procedure models, the data model, and their supporting dictionaries are automatically updated where necessary.

## Section D
## Rapid Application Development (RAD)

**Section Overview**

Rapid Application Development (RAD) is a development technique that enables organizations to respond quickly when the requirements are time-sensitive and mission-critical. RAD is effective when the life cycle used is optimized for high-speed development.

**Definition**

Rapid Application Development (RAD) is a calendar-driven approach to development that imposes limits on the time available for specifying requirements, building prototypes, and constructing applications.

It encourages users to:

- Prioritize requirements
- Establish and recognize scheduling and resource limitations.

**In This Section**

## Overview

**RAD Techniques**

RAD uses a combination of five mechanisms/techniques to accomplish the goal of meeting the requirements quickly. Used together with a well-defined and planned methodology, these techniques allow applications to be built much faster while maintaining quality.

Each is discussed below.

**Small Teams/Mentors**

One key to fast development is using experienced small teams and/or individual mentors that have received intensive training in the use of CASE and code generation tools and supporting techniques and methodologies.

**Prototyping**

Prototypes built prior to the RAD effort may be used to clearly communicate the basic requirements of the application. In some cases, output created by the prototype may serve as initial input into the RAD effort.

Prototype details are successively refined until sufficient design information has been accumulated to generate the code for the application.

**Reference:** Volume I, Chapter 11, Section A, *Prototyping*

**CASE Tools**

CASE tools may perform a number of functions in the RAD environment:

- Process and data modeling
- Structure charting
- Code generation
- Generation of physical databases
- Generation of documentation and project management data
- Storage of design information in a central repository

**Note:** Access to a CASE tool with a repository feature is critical to realizing the full value of using RAD. This technique will be less effective until the repository is populated with previous models as it relies heavily on re-use.

**Reference:** Volume I, Chapter 11, Section C, *CASE Tools*

*Continued on next page*

## Overview, Continued

**Joint Application Development**

The communication of specifications from users to analysts may be facilitated in RAD through the use of Joint Application Development (JAD) workshops. RAD uses a CASE repository to capture the requirements and design information generated during a JAD session.

**Reference:** Volume I, Chapter 11, Section B, *Joint Application Development*

**Prioritization**

Users must prioritize requirements and recognize that getting the application into production on schedule is the primary goal. The priorities established by the users dictate the order of the functional development and deployment.

There is no time in the schedule for “scope creep”.

## Benefits of RAD

**Overview of Benefits**

There are four major benefits to using RAD:

- Coordination
- Time Considerations
- Defect Elimination
- Consistency

Each benefit is discussed in greater detail below.

**Coordination**

The RAD development technique, in conjunction with an Incremental life cycle, encourages the review and coordination of subsystem functional requirements with the end users in a timely fashion.

User knowledge and experience are incorporated through the use of JAD workshops. User needs are met more effectively because they are continuously involved in the development process.

**Reference:** Volume I, Chapter 11, Section B, *Joint Application Development*

**Time Considerations**

RAD may allow the SDLC life cycle phases to overlap, thereby shrinking the total development time of the project.

**Caveat:** In order to reap the full benefit of phase overlap, the project's development process must be well-planned before RAD is initiated.

**Defect Elimination**

The iterative process of RAD encourages early defect detection. This allows for efficient maintenance and change support.

Code is automatically generated from the design as much as possible, thereby minimizing the amount of human-generated code and the potential for human error.

**Consistency**

All design specifications are stored in the repository and extensively re-used, thereby providing consistency within and between applications.

*Continued on next page*

## Benefits of RAD, Continued

**Object-Oriented Technology**

RAD may be useful in an Object-Oriented environment because:

- RAD encourages heavy re-use of the models which exist in the repository.
- RAD takes advantage of the data residing in an integrated dictionary.

**Reference:** Volume I, Chapter 11, Section E, *Object-Oriented (OO) Technology*

## Risks of RAD

**Who Should Decide?**

The decision to use RAD should **not** be user-driven. Appropriate management and development team personnel must make an initial evaluation to determine if a proposed system is a candidate for this technique.

**Management Support**

To use RAD successfully, it is critical that full management support and commitment is available to remove obstacles to completing each cycle on time. This commitment must include all resources (personnel, funding for overtime, computer resources, etc.) necessary to meet the established schedule.

**User Commitment**

Extensive involvement and commitment from the user is required. If the user is unwilling or unable to commit the resources (knowledgeable personnel and time) necessary for timely inputs and reviews, RAD should not be used.

**Project Control**

The following project control issues must be addressed:

- Change Control: The scope of requirements for each release should be clearly delimited to avoid “scope creep”.
- Because the number of iterations and changes is unspecified at the beginning of the project, a RAD effort can be hard to manage.
- All documentation must be clearly completed and updated as required during each RAD effort, even if re-work will be needed later.

## Life Cycle Considerations

**Life Cycle Impact**

When using RAD, the life cycle phases are combined and overlapped, yet baselines must be established and maintained.

**Note:** The phases do not change from a standard Waterfall Life Cycle except in order of completion.

**Project Initiation Phase**

Most RAD projects will be considered high-risk projects due to short delivery cycles and potentials for complexity.

A feasibility study is completed as part of the Cost/Benefit Analysis. If the Business Sponsor agrees, the Project Plan is then developed by the Project Initiation Team. The plan should include:

- The declaration that this is to be a RAD project
- A list of System Development Team members
- Other pertinent tailoring information

The project schedule should specify the amount of time allotted to accommodate reworks or go-backs to earlier stages as more attractive alternatives are identified, new issues arise, or priorities change.

**Reference:** Volume I, Chapter 6, *Project Initiation*.

**Project Definition Phase**

The users must set their priorities and determine which requirements will be implemented in the initial release of the application. The System Development Team works closely with the users to build an understanding of the total system requirements, individual and team capabilities, the tools to be used, and the time frame.

Detailed elaboration of requirements may be deferred until later phases. As the software is built, it is expected that design options will stabilize as standards are developed and invoked.

*Continued on next page*

## Life Cycle Considerations, Continued

**Project Definition Phase** (continued)

The Baseline Definition is completed only upon agreement of the Systems Development Team and the Business Sponsor. Further changes will be accepted as Change Requests and will be dealt with by the Change Control Board.

**Reference:** Volume II, Chapter 7, Section B, *Project Definition*.

**System Design Phase**

The System Design Phase loosely follows the requirements set in the Definition Phase. In projects using new technology, the design is done concurrently with the development and coding phases.

It is expected that most of the rework and go-backs will occur during this phase. Frequent quality assurance reviews such as design reviews and peer reviews/code walkthroughs are helpful in gaining the most value from RAD.

**References:**

- Volume II, Chapter 7, Section C, *System Design*
- Volume I, Chapter 3, Section C, *Technical Reviews*

*Continued on next page*

## Life Cycle Considerations, Continued

**Programming or Construction Phase**

Coding for the application may be done concurrently with the System Design Phase, particularly when new technology is introduced.

| Role | Responsibilities |
|---|---|
| System Development Team | Perform the unit and integration testing of the system under construction. |
| User | • The users must understand and acknowledge that there will be several releases until all requirements are met. This should be documented in the Project Plan.<br><br>• If the user feels that there is a misunderstanding of the functional requirements, they must take this to the Systems Development Team immediately. Failure to do so may delay product delivery. |

**Reference:** Volume II, Chapter 7, Section D, *Programming or Construction*

**Implementation or Transition Phase**

Initial implementation (i.e., piloting) will be done on the testing platform. Lessons learned from each release will be documented and shared with the group responsible for implementation.

Requirements, design, security, and system documents are updated during each release.

Production implementation occurs after all development associated with an individual release is complete and the Operational Readiness Review for the individual release has been conducted.

**Reference:** Volume II, Chapter 7, Section F, *Implementation or Transition*

**Operations Phase**

Maintenance may be done by a group other than the System Development Team. Therefore, it is imperative that all documentation be complete.

## Section E
## Object-Oriented (OO) Technology

**Description**

Object-Oriented (OO) technology:

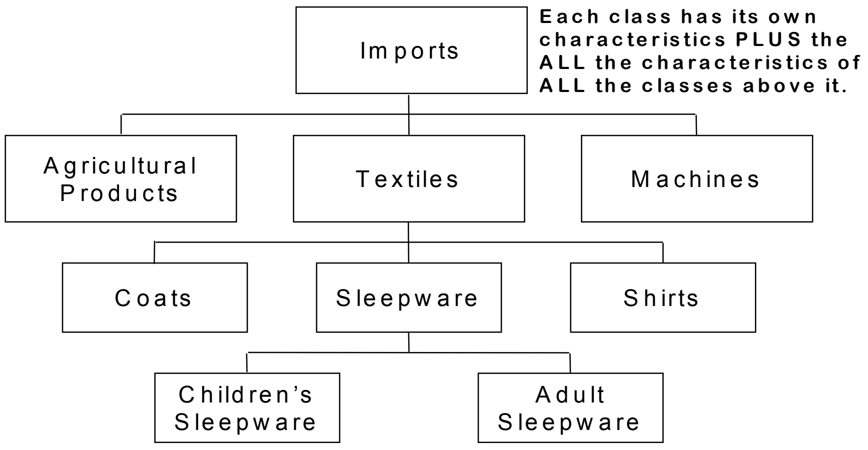
- Is an approach to software construction that (in the long run) may solve some of the classic problems associated with the increasing complexity of large-scale software development and maintenance
- A technique for system modeling -- either a software system or a system in a wider context: software and hardware

**Definitions**

**Object:** A bundle of data (also known as the attributes of the object) and operations (also known as its methods) that can act on the data. All objects have identity and are distinguishable.

**Class:** A template for multiple objects describing how the objects are structured internally.

**Inheritance:** A relationship among classes, wherein one class shares the structure or behavior defined in one (single inheritance) or more (multiple inheritance) other classes. This leads to a hierarchical structure of "parent" or superclasses, each with one or more "children" or subclasses below them.


Imports
Each class has its own characteristics PLUS the ALL the characteristics of ALL the classes above it.
Agricultural Products
Textiles
Machines
Coats
Sleepware
Shirts
Children's Sleepware
Adult Sleepware

*Continued on next page*

## Object-Oriented (OO) Technology, Continued

**Life Cycle**

Because object technology is built on the idea of layers and clusters of interrelated function groups, it may lend itself most easily to the use of an Incremental Life Cycle or an Information Engineering Life Cycle.

**References:**

- Volume II, Chapter 8, *Incremental Life Cycle*
- Volume II, Chapter 9, *Information Engineering Life Cycle*

**Core Concept**

The core concept of OO technology is defining the world in the realm of objects and putting like objects together into classes. These objects can then be translated into code and the relationships between objects are better understood.

**In This Section**

## Advantages and Disadvantages of OO

**Advantages**

The advantages of OO technology include:

- Once a stable object library is in place, object reuse allows software to be developed in a fraction of the time and cost of conventional methods.
- Object technology may result in systems which better solve the business problem that they address.
- In many cases, individual objects may be modified without significantly affecting other objects. This makes the software easier to modify and maintain. It also allows the software system to grow and evolve in response to changing circumstances.

**Disadvantages**

A few of the disadvantages of OO technology are:

- Up-front costs to institute OO (or any other new technology) in an organization are substantial. Costs may include:
  - Development Platforms
  - Tools
  - Training
  - The time required to develop a stable library of reusable objects
- Without the use of software development tools, documentation/tracing requirements are difficult to meet and have not been thoroughly incorporated into any OO methodology at this time.
- The test preparation (i.e., development of test scenarios, test specifications) may be more difficult to do if there is a lack of a clear method or tool for documenting requirements.

## Performance Risks

**Introduction**

There are a number of performance risks associated with using OO technology in a mainframe environment. Any one of these may have an adverse impact on system performance.

**Messaging**

**Definition:** An operation that one object performs on another.

Compared to a procedural language, there is a definite performance cost for sending a message from one object to another. In the worst case, a message may take nearly three times as long as a simple subprogram call.

**Cascade Effect**

OO systems are developed in layers. Individual functional groupings are generally very small, since they build on lower-level functional groupings. However, this has a cascade effect in time required as messaging moves down through the layers. For applications in which time is a limited resource, messaging may be unacceptable.

**Mainframe Paging and Swapping**

Most mainframe compilers allocate object code in segments, with the code for each compilation unit placed in one or more segments. This model assumes a high degree of co-location; subprograms within one segment call subprograms within the same segment. However, in OO systems, there is rarely such co-location. This may lead to thrashing during execution.

**Encumbrance of Classes**

If a class is at the bottom of the hierarchical structure, it may have many superclasses, whose code must be included when linking to the most specific class.

**Allocation of Objects**

Allocating an object is a dynamic action that uses more computing cycles because of the extra commands that must be issued to manage processor memory resources. For some time-critical applications, one cannot afford the extra cycles needed to manage allocated objects.

## Documentation Considerations

**OO Project Requirements**

In addition to the regular project documentation, the System Development Team will develop a Style Guide as one of the first activities of an OO project.

Also, Use Cases and Event Diagrams are required to be developed by projects using the OO methodology to clarify definitions for requirements and objects.

Due to the lack of sufficient requirements documentation in the OO methodology, the System Development Team will also develop, document, and use a mechanism to track the requirements supported by the use cases and event diagrams. This mechanism to track requirements will also aid in the formal and informal testing of the system being developed.

**Style Guide**

The Style Guide will:

- Show the manner in which the objects and classes will be represented
- Show how the requirements will be documented for use by the testers
- Identify any tailoring of the requirements document.

The style selected will be dependent upon the availability of a tool for the OO development effort.

**Use Case**

**Definition:** A special sequence of transactions in a dialog between a user and the system. A given use case denotes some function or business process.

Use Case shall be the mechanism for documenting requirements. Use Cases shall include:

- Use Case name
- Short Description
- Diagram Scenario
- Assumptions
- Referenced Use Cases
- Issues/concerns
- Additional documentation which relates to the use case

*Continued on next page*

## Documentation Considerations, Continued

**Event Diagrams**

**Definition:** Event diagrams document the process of changing the states (events or status) of the object. Event diagrams show the states of an object and how one acts on an object to move it to another state. Event diagrams can also be used as a supplement to document requirements.

Event diagrams consist of:

- **Operation:** An action that one object performs upon another. The terms method, and operation are usually interchangeable.
- **Event:** Some occurrence that may cause the state of a system to change
- **Control Condition:** A statement which identifies under what conditions an operation is permitted to begin.
- **Trigger:** An action that causes an operation to begin.

## Configuration Management Considerations

**Overview**

The long range economies inherent in OO technology are based on the concept of reusing previously developed objects. In order to do this efficiently, configuration management becomes a vital function within the OO project.

However, many of the tools being used today for OO development have little configuration management functionality.

**Object Librarian**

One individual on the project shall be assigned responsibility for controlling and managing the object library.

- Strict configuration controls shall be placed on this library so that everyone knows which model is the current baseline.
- If an object modeling tool is used, procedures will be developed and documented to handle copies of the same model when an Incremental Life Cycle is used.
- The object librarian can be either a Customs employee or a contractor, as long as a detailed process is in place and documented to address the care and feeding of the object library.

**Baselines**

There are two parts to the product baseline for an OO project:

- Functional Baseline
- Allocated Baseline

**Both of these baselines are mandatory deliverables for each project or release of a system using OO technology.** These baselines are discussed below.

*Continued on next page*

## Configuration Management Considerations, Continued

**Functional Baseline**

**Components:** The functional baseline consists of :

- Use Case Diagrams
- High Level Diagrams
- Requirements

**Configuration Management Requirement:** If an Incremental Life Cycle is chosen for the project, there may be several versions of the use case diagrams and the event diagrams. There must be strict controls placed on these items, so that when a new release is being developed, there is a baseline of the previous version of the use case and event diagrams and a mechanism for identifying the changes from one release to the next.

**Requirements:** The requirements associated with the use cases need to be documented and baselined at the same time as the use case to ensure consistency.

**Models:** The model associated with the use cases should also be baselined and strictly controlled.

**Allocated Baseline**

**Components:** The allocated baseline consists of:

- **Business Object Model:** A model which describes the static structure of the objects in the business environment and the relationship(s) among the objects

- **System Component Model:** A model of the parts which make up the system

- **Object Model Design Document:** This document describes the object model and the data associated with the object model.

- **System Modules and Programs:** The software modules and programs which make up the system

**Usage:** The allocated baseline will be used for unit, integration, system, acceptance, and operational testing.